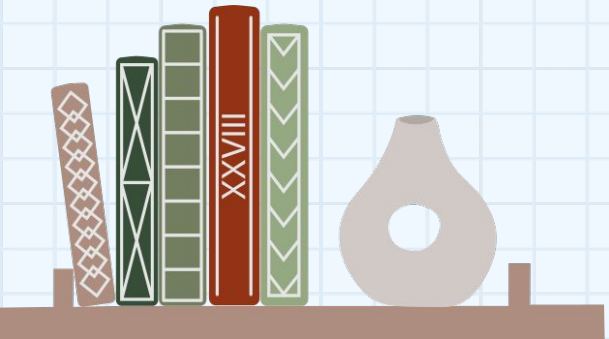
Converting Characters to Numbers

Using ord() and Raw Methods



Introduction

Characters are stored as numbers internally (ASCII/Unicode).

Converting characters to numbers is useful in **encryption, data processing, and numerical operations**.

Example:

`'A'` → `65`

`'1'` → `49`

`'@'` → `64`

Explanation:

- Computers represent characters as numbers based on **ASCII** or **Unicode**.
- Python provides **built-in functions** and **manual methods** for conversion.

The ord() Function

- `ord()` converts a character into its **ASCII/Unicode integer value**.

- **Syntax:**

```
ord('A')  # Output: 65
ord('1')  # Output: 49
ord('z')  # Output: 122
```

Explanation:

- `ord('A')` returns `65`, which is the ASCII code of `'A'`.
- Similarly for ‘1’ and ‘z’
- This function is useful for numerical operations on characters.

Converting String to Numbers

- Convert each character of a string to numbers using `ord()`.

Example(code):

```
text = "Hello"
ascii_values = [ord(c) for c in text]
print(ascii_values)  # Output: [72, 101, 108, 108, 111]
```

Explanation:

- Each character in `"Hello"` is converted to its ASCII value.
- **List comprehension** makes this conversion easy.

Using Loops for Conversion

- Convert characters to numbers manually using a **loop**.

Example (code):

```
text = "Python"
for char in text:
    print(f"Character: {char}, ASCII: {ord(char)}")
#output
Character: P, ASCII: 80
Character: y, ASCII: 121
Character: t, ASCII: 116
Character: h, ASCII: 104
Character: o, ASCII: 111
Character: n, ASCII: 110
```

Explanation:

- This method **iterates** through each character and prints its ASCII value.

Summing ASCII Values of a String

- Compute the sum of ASCII values of all characters.

Example:

```
text = "abc"
ascii_sum = sum(ord(c) for c in text)
print("Total ASCII Sum:", ascii_sum)  # Output: Total ASCII Sum: 294
```

Explanation:

- The ASCII values of `'a'`, `'b'`, and `'c'` are summed (`97 + 98 + 99 = 294`).
- This technique is useful for **checksums and encryption**.

Extracting Digits from a String

- Convert numeric characters to actual integers manually.

Example:

```
text = "A1B2C3"
numbers = [ord(c) - ord('0') for c in text if '0' <= c <= '9']
print(numbers)  # Output: [1, 2, 3]
```

Explanation:

- This extracts digits (`'1'`, `'2'`, `'3'`) from `"A1B2C3"`.
- **Formula:** `ord(c) - ord('0')` converts `'1'` to `1`, `'2'` to `2`, etc.

Converting a Whole Number String to Integer

- Convert `"1234"` (as a string) into an actual number manually.

Example (using loops):

```
def string_to_int(s):
    num = 0
    for char in s:
        num = num * 10 + (ord(char) - ord('0'))
    return num

print(string_to_int("1234"))  # Output: 1234
```

Explanation:

- Each digit is **processed one by one**, multiplying the result by 10 and adding the next digit.
- Works similar to `atoi()`.

Handling Non-Numeric Characters in Conversion

- What if the string contains non-numeric characters?

Example (safe conversion):

```
def safe_string_to_int(s):
    num = 0
    for char in s:
        if '0' <= char <= '9':  # Only process digits
            num = num * 10 + (ord(char) - ord('0'))
        else:
            return "Invalid Input"
    return num
print(safe_string_to_int("567"))    # Output: 567
print(safe_string_to_int("56a7"))   # Output: Invalid Input
```

Explanation:

- **Checks** if the character is a digit before processing.
- Returns `"Invalid Input"` if it finds non-numeric characters.

Introducing atoi() (C/C++ Equivalent)

- In **C/C++**, a similar function called `atoi()` (ASCII to Integer) exists.

Example in C :

- Similar **manual methods** exist in C/C++ using loops.

```c
#include <stdlib.h>
int num = atoi("1234");  // Output: 1234
```

Explanation:

- Python's **manual conversion** methods mimic `atoi()` from C/C++.
- **Python provides `int()`**, which is more powerful than `atoi()`, but raw methods are useful for learning.

Problems with atoi()

- **No error handling** → `"abc"` returns `0`
- **Cannot detect large numbers (overflow)**
- **Only works with whole numbers**

Better Alternatives

1. `strtol()` → Handles errors & large numbers
2. **C++** `stoi()` → Safer alternative in C++

Example using `strtol()`:

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    char str[] = "456";
    char *endptr;
    long num = strtol(str, &endptr, 10);

    printf("Number: %ld\n", num);
    return 0;
}
```

Output

```
Number: 456
```

Explanation:

- `strtol()` provides better error handling by pointing to invalid characters in the string.
- `stoi()` is a modern C++ alternative with exception handling.

Summary

- `ord()` converts characters to ASCII values.
- **Loops & formulas** can be used for manual conversions.
- **Handling errors** is important for safe conversion.
- `atoi()` in C/C++ works similarly.

Thank You